Portable Monte Carlo Transport Performance Evaluation in the PATMOS Prototype

T. Chang, E. Brun, C. Calvin

Portable Monte Carlo Transport Performance Evaluation in the PATMOS Prototype

Tao CHANG[1], Emeric BRUN[1], and Christophe CALVIN[2]

[1] DEN-Service d'Etudes des Réacteurs et de Mathématiques Appliquées (SERMA)
[2] CEA DRF, Maison de la Simulation
CEA, Université Paris-Saclay, F-91191, Gif-sur-Yvette, France
`{tao.chang,emeric.brun,christophe.calvin}@cea.fr`

Abstract. A heterogeneous offload version of Monte Carlo neutron transport has been developed in the framework of PATMOS prototype via several programming models (OpenMP thread, OpenMP offload, OpenACC and CUDA). Two algorithms are implemented, including both history-based method and pseudo event-based method. A performance evaluation has been carried out with a representative benchmark, slabAllNuclides. Numerical results illustrate the promising gain in performance for our heterogeneous offload MC code. These results demonstrate that pseudo event-based approach outperforms history-based approach significantly. Furthermore, by using pseudo event-based method, the OpenACC version is competitive enough, obtaining at least 71% performance comparing to the CUDA version, wherein the OpenMP offload version renders low performance for both approaches.

Keywords: Monte Carlo transport, history-based method, pseudo event-based method, OpenMP thread, OpenMP offload, OpenACC, CUDA

1 Introduction

Monte Carlo (MC) neutron transport simulation is a stochastic method that is widely used in the nuclear field to perform reference calculations. Instead of solving the radiation transport equation by introducing discretizations and physical approximations, the MC method simulates the life of a large number of particles from their birth to their death. Their life consists in a succession of random flights and collisions. From the caracheristics of these events, physical quantities can be computed such as the density of particles, the reaction rates, the heat power. The very few approximations introduced make the MC method a precise approach of neutron transport simulation under complex conditions. However it incurs a much higher computational cost comparing to the deterministic method used in the industry.

For this reason, researchers and developers of many MC transport solvers have turned to solutions by porting MC codes to modern architectures. In order to achieve performance, the vectorization of MC codes comes to be a major aspect to dig into. However, the conventional MC algorithm, also known as the history-based method is considered as an embarassingly parallel algorithm wherein particles are being simulated independently during their lifetime. This main feature makes it an unsatisfactory candidate for data-level parallelism.

As an alternative solution, the event-based method was proposed by regrouping particles according to their different event types (absorption, collision, migration, scattering) and undertaking the simulation of banked particles in parallel [1]. This approach is better suited for vectorization on modern computers, while it introduces extra workload of consolidating surviving particles during the simulation. Furthermore, event-based method requires restructuring the control flow and redesigning data structure, making it difficult to implement with full-physics capabilities.

Recently, a number of studies have explored using Intel MIC and Nvidia GPU for MC transport solvers and micro-benchmarks [2–5]. Both history-based method and event-based method are implemented and the results are quite informative and promising. They have proved that history-based method is far more straightforward to implement than event-based method and event-based method may outperform history-based method with specific tuning strategies such as remapping data references, use of intrinsic functions and design of trivial kernel. However, all these work did not take into account the portability and the performance portability of MC codes, which are two significant factors to consider for software development. To our knowledge, the only work related to the implementation of MC solver with concern of portability is carried on by Bleile and his group [6] where they developed a portable event-based MC solver relying on the Nvidia Thrust library and discovered that Thrust version can only obtain a maximum of 36% performance comparing to CUDA version and this percentage keeps decreasing while increasing the number of particles. The investigation highlights the trade-off between portability and performance for MC solvers and shows the lack of optimizations for portable codes.

The objective of this paper is indeed to address the challenge above by performing a performance evaluation of exisiting programming models in the framework of MC neutron transport codes on modern architectures (CPU + GPU). The rest of the paper is organized as follows. Section 2 gives a brief introduction of MC neutron transport application, benchmark as well as programming models that we use for experiments. In Section 3, MC implementations are described in detail in terms of algorithms and programming languages. Section 4 covers the comparison of performance based on different algorithms and architectures. Several concluding remarks are drawn in Section 5, as well as certain plans for future development.

2 Background

2.1 The PATMOS Monte Carlo Prototype

PATMOS is a prototype of Monte Carlo neutron transport under development at CEA dedicated to the testing of algorithms for high-performance computations on modern architectures. It relies on a hybrid parallelism based on MPI for distributed memory and OpenMP or C++ native threads for shared memory. One of the goals is to perform pin-by-pin full core depletion calculations for large nuclear power reactors with realistic temperature fields. PATMOS is entirely written in C++, with a heavy use of polymorphism in order to always allow the choice between competing algorithms such as the mix of nuclides with pre-computed Doppler-broadened cross sections and on-the-fly Doppler broadening [7].

The physics of PATMOS is simplified with two types of particles (monokinetic pseudo-particles and neutrons). Four types of physical interactions including elastic scattering, discrete inelastic scattering, absorptions and fission have been implemented. The scoring part is encapsulated into a scorer class which deals with tally computation during the simulation and gathers statistical results afterwards.

2.2 Benchmark

To evaluate performance of different programming models in the framework of PATMOS, a benchmark named slabAllNuclides was implemented to perform a fixed source MC simulation with SIGMA1 Doppler Broadening method [9] using a slab geometry with an arbitrary number of heterogeneous regions (10000 volumes). Each material contains 388 nuclides of the ENDFBVIIr0 library at 900K. The main components of the mixture are H1 and U238 so as to obtain a representative Pressurized Water Reactor (PWR) spectrum.

On one hand, Doppler Broadening introduces compute-intensive FLOP work between frequent memory loads to mitigate the latency-bound bottleneck mainly induced by the binary search in the pre-tabulated cross section approach [10]. On the other hand, temperature dependent cross section data are computed on-the-fly whenever they are requested, which saves significantly the memory footprint of program. Previous work done by Y. Wang [11] has also proved that there are few opportunities to exploit vectorization for Monte Carlo algorithms based on pre-tabulated cross section. All these facts make us choosing on-the-fly Doppler Broadening to fully explore performance on "memory-limited" architectures.

2.3 Programming Models

PATMOS allows two levels of parallelism via MPI + Multi-thread Libraries. In order to address architectures where multiple accelerators are associated with single node, we can use either "Multiple Threads, Single Accelerator (MTSA)" or "Multiple Threads, Multiple Accelerators (MTMA)" strategies. The main difference between them is that MTSA requires one MPI process using only one accelerator while MTMA allows multiple shared memory threads to target multiple accelerators. MTMA avoids extra launch latency, which makes it a better strategy for Monte Carlo simulations on heterogeneous systems. Thus, we carried out a set of intra-node experiments using test cases mentioned above with a simple hybrid programming model OpenMP thread + $\{X\}$, where $\{X\}$ can be any languages which are capable of parallel programming on modern accelerators. The programming languages that we have used for implementation are listed below:

1. *CUDA*: A low-level programming language created by Nvidia that leverages Nvidia GPUs to solve complex computational problems [12]. It allows developers to interoperate with C, C++, Fortran, Python or other languages, which provides an efficient interface to manipulate Nvidia GPUs in parallel programming. To explore maximal computing power of Nvidia GPU architectures, one shall take advantage of CUDA thread and memory hierarchies.
2. *OpenACC*: A user-driven performance-portable accelerator programming language supporting implicit offload computing by directives [13]. The directives can be used to accomplish data transfer, kernel execution and stream synchronization.
3. *OpenMP offload*: From OpenMP 4.0, the specification starts to provide a set of directives to instruct the compiler and runtime to do offload computing targeting to devices such as GPUs, FPGAs, Intel MIC, etc.

CUDA provides three key abstractions to explore maximal computing power of GPU architectures (thread hierarchy, memory hierarchy and barrier synchronization). From the thread hierarchy perspective, CUDA makes use of three levels of work units to describe a `block-warp-thread` parallelism. From the memory hierarchy point of view, CUDA exposes a group of programmable memory types such as registers, shared memory, constant memory, and global memory. Concerning barrier synchronization, CUDA provides system-level and block-level of barrier synchronization.

As a comparision, OpenACC and OpenMP offload both offer a set of directives to express thread hierarchy (OpenACC: `gang-worker-vector`, OpenMP offload: `team-parallel-simd`) whereas they do not offer programming interface to on-chip memory and thread synchronization. The lack of access to entire CUDA's feature set may lead to a considerable penalty of performance for OpenACC and OpenMP offload [14]. The key difference between OpenACC and OpenMP offload is that OpenACC supports CUDA asynchronous multistreaming with the directive `async(stream_id)` while OpenMP offload offers `nowait` clause which poorly performs this functionality.

3 Implementations

We begin the discussion of implementations in PATMOS by performing a CPU-based slabAllNuclides test in terms of runtime percentage. All results were retrieved by C++ native clock and perf.

In Table 1, `binary_search`, `compute_integral` and `buildMedium` are user-defined functions for the calculation of cross section and the initialization of simulation. `erfc` and `exp` are the most consuming mathematical functions to calculate the complementary error and the base e exponential which are needed for SIGMA1 Doppler Broadening.

Table 1. slabAllNuclides runtime percentage

Processing Step	**Runtime Percentage** (%)
Total Cross Section	95.4
`exp`	17.6
`erfc`	49.4
`binary_search`	2.4
`compute_integral`	79.2
Partial Cross Section	1.7
`erfc`	0.6
`compute_integral`	1.4
Initialization	1.8
`buildMedium`	1.5

Because the total cross section calculations account for up to 95% of total runtime, we decided to offload GPU version which only caculates microscopic total cross sections on device. All other parts of MC simulations are executed on host, which significantly reduces the workflow of device codes and increases the possiblity for performance of OpenACC and OpenMP offload versions to match up to the performance of CUDA version.

Algorithm 1 shows the procedure of slabAllNuclides benchmark using history-based method. Miscroscopic cross section lookup is the only part offloaded to device. Once the required data for cross section lookups are transfered from host to device, calculations on device begin and the host summarizes macroscopic total cross section after the results being transfered back. It is obvious that our design brings in too many back-and-forth data transfers between host and device. The size of data movement for each calculation (a group of nuclides in one material) is quite small but the large number of memcpy calls induces many launch overheads which may degrade performance in an overwhelming way.

Algorithm 1: History-based algorithm

```
foreach particle generated from source do
    while particle is alive do
        calculation of macroscopic cross section:
            • do microscopic cross section lookups $\Longrightarrow$ offloaded;
            • sum up total cross section;
        sample distance to collision in material;
        if new position is still inside material then
            move particle to new position, and do collision;
        else
            move particle across boundary;
        end
    end
end
```

Algorithm 2: Pseudo event-based algorithm

```
1  foreach bank of N particles generated from source do
2      while particles remain in bank do
3          foreach remaining particle in bank do
4              bank required data for microscopic cross section lookups;
5          end
6          do microscopic cross section lookups $\Longrightarrow$ offloaded;
7          foreach remaining particle in bank do
8              sum up total cross section;
9              sample distance to collision in material;
10             move particle and do collision;
11         end
12     end
13 end
```

Algorithm 3: Microscopic cross section lookup

```
Input: randomly sampled a group of $N$ tuples of materials, energies and
       temperatures, $\{(m_i, E_i, T_i)\}_{i \in N}$
Result: caculated microscopic cross sections for $N$ materials, $\{\sigma_{ik}\}_{i \in N, k \in |m_i|}$
1 block - gang - team level
2 for ($n_{ik}$, $E_i$, $T_i$) where $n_{ik} \in m_i$ do
3     $\sigma_{ik} = pre\_calcul()$;
4     thread - vector level
5     foreach thread in warp do
6         $\sigma_{ik} \mathrel{+}= compute\_integral()$;
7     end
8 end
```

We managed to use zero-copy pinned memory to mitigate this bottleneck for the CUDA version, whereas the OpenACC and OpenMP offload versions have no support for such technique. As an alternative solution, we also tried to merge multiple small kernels into a big one by grouping a set of calculations for different particles together. This is achieved by banking multiple particles into one group and offloading microscopic cross section lookups for all these particles. In this way, the number of data transfers can be reduced and the amount of work for each kernel is increased.

To fulfill this tuning strategy, our history-based method is redesigned to a new "pseudo event-based" approach. The details of this new method are explicitly described in Algorithm 2. The overall procedure of history tracking is reorganized into two `for` loops, where the first loop stores all required data for calculation of microscopic cross sections of all alive particles and the second loop takes the responsibility to do particle movement and interaction. Between two loops, the microscopic cross section lookups are executed host or device.

Algorithm 3 shows the algorithm of microscopic cross section lookup using CUDA thread hierarchy. In contrast to the pre-tabulated offloading version, which usually uses 1 thread to calculating microscopic cross section of 1 nuclide, we decided to use 32 threads (a warp) consuming 1 nuclide because the on-the-fly Doppler Broadening computation injects an inner-loop for integral calculation.

4 Results

To evaluate performance of programming models, we carried out a series of intra-node tests on two architectures listed as follows:

• **Ouessant:**	2× 10-core IBM Power8, SMT8	+ 4× Nvidia P100
• **Cobalt-hybrid:**	2× 14-core Intel Xeon E5-2680 v4, HT	+ 2× Nvidia P100

On Ouessant, PATMOS was compiled with GCC 7.1, CLANG 3.8.0, PGI 18.10, and XLC 16.1.1 combined with CUDA 9.2. On Cobalt-hybrid, GCC 7.1, CLANG 5.0, PGI 18.7 and Intel compiler 17.0 were used with CUDA 9.0. The input parameters of slabAllNuclides are fixed to 2×10^4 particles, 10 cycles for the following tests. The bank size of pseudo event-based method is set to 100.

We firstly performed a CPU test to obtain the baseline performance on each architecture. We find that history-based method and pseudo event-based method make no difference on CPU. It turns out that the peak performances of slabAll-Nuclides are 4.7×10^2 particles/s (PGI, 20 cores, SMT8) and 12.1×10^2 particles/s (Intel compiler, 28 cores, HT2) on Ouessant and Cobalt-hybrid.

The performance gap between two machines is produced by different level of vectorization achieved by compilers. Intel compiler automatically takes advantage of processing power of Intel architecture and leads to performance improvement comparing to other compilers (2.6× speedup).

4.1 Programming Model Performance Evaluation

A series of tests were carried out either on CPU or on GPU so as to evaluate the performance of the different programming models. Numerical results of peak performance are illustrated in Table 2.

We find that among the implemented programming models, the CUDA and OpenACC versions allow to obtain better performance (at maximal 66×10^2 and 52×10^2 particles/s on Ouessant, 63×10^4 and 45×10^4 particles/s on Cobalt-hybrid). The OpenMP offload version is not competitive with the OpenACC version, merely leading to a tracking rate of 12×10^2 particles/s on Ouessant.

Figure 1 provides a straightforward view for comparision of performance speedup among different programming models and architectures. They reflect that (1) the performance of slabAllNuclides running on Cobalt-hybrid is better than the performance running on Ouessant. (2) Pseudo event-based method (*PEB*) outperforms history-based method (*HB*) with a factor of $3 - 6$ speedup.

Table 2. Particle tracking rate via different programming models

Machine		Programming Model	slabAllNuclides ($\times 10^2$ particles/s) HB	PEB(100)
Ouessant	CPU (20 cores, SMT8)	OMPth	4.7	4.7
		OMPth+ACC	4.6	4.5
		OMPth+offload	3.7	3.7
	4P100	OMPth+CUDA	23.7	65.8
		OMPth+ACC	9.4	52.4
		OMPth+offload	5.0	12.2
Cobalt-hybrid	CPU (28 cores, HT2)	OMPth	12.1	12.1
		OMPth+ACC	5.6	5.0
	2P100	OMPth+CUDA	17.0	62.5
		OMPth+ACC	7.7	44.6

where `OMPth` refers to OpenMP thread, `OMPth+offload` means OpenMP host and offload functionalities, `ACC` is equal to OpenACC.

(3) The OpenMP offload version cannot be compiled on Cobalt-hybrid, it renders much lower performance comparing to the CUDA and OpenACC versions.

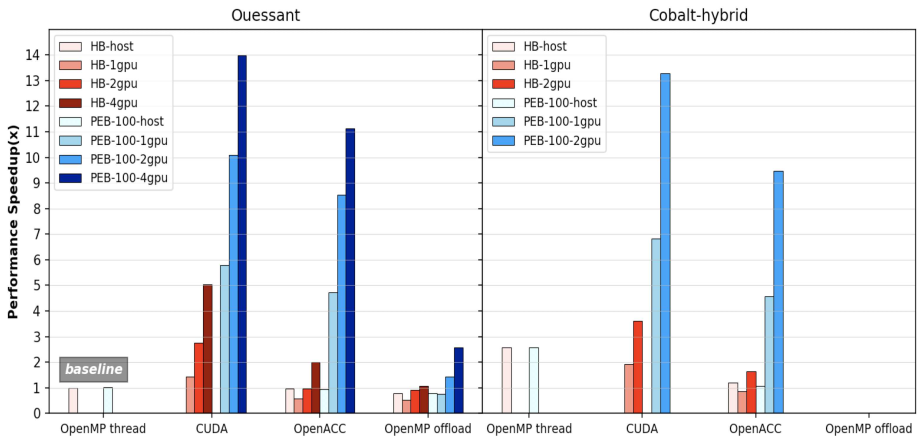

Fig. 1. slabAllNuclides performance speedup - Baseline for the performance is obtained on Ouessant with 20 cores, SMT8

Overall, it is obvious that on GPU, pseudo event-based method is more suitable than history-based method. The OpenACC version is competitive with CUDA version via pseudo event-based method. The OpenMP offload version provokes a huge performance degradation via either history-based method or pseudo event-based method. This performance penalty is caused by underdeveloped support of CUDA streams. As for CPU, on Power architecture the performances of OpenACC and OpenMP offload versions are close to that of OpenMP thread version because all compilers achieve the same level of vectorization for SIGMA1 Doppler broadening algorithm with 128-bit vector register (VSX). However, on x86 architecture, the best performance is obtained by Intel C++ compiler which manages to vectorize the loop with 256-bit vector register (AVX2). The OpenACC version introduces a performance penalty because it is compiled with PGI that does not vectorize the main loop of SIGMA1 algorithm.

5 Conclusion

This paper depicts the implementation of a PATMOS benchmark slabAllNuclides in the framework of MC neutron transport via different programming languages (OpenMP threads, OpenMP offload, OpenACC). The total microscopic cross section lookup is the unique part which is offloaded to accelerators. This implementation is the first study to use OpenACC and OpenMP offloading functionality for MC simulation and to offer comparisons between programming models in terms of performance. We describe an alternative algorithm "pseudo event-based method" for the purpose of mitigating performance bottleneck incited by conventional "history-based" method.

Performance results are provided across two computing architectures (Ouessant and Cobalt-hybrid). It is clear that the GPU performance via pseudo event-based method surpasses significantly history-based method. The OpenACC version can obtain at least 71% CUDA performance with pseudo event-based method. In contrast, the one with history-based method is limited to only 45% of the CUDA version. With respect to the OpenMP offload version, both history-based method and pseudo event-based method can only exploit around 20% CUDA performance which is completely unsatisfactory. We can conclude that (1) OpenACC is a good choice for the development of portable pseudo event-based MC simulation in the context of our heterogeneous offload strategy. (2) OpenMP offload is not suitable for CPU threads + GPU model since the underdeveloped support of CUDA asynchronous stream restrains the parallelism on GPU side.

There are several capabilities that we intend to implement for future development. From MC simulation side, since we have demonstrated that porting total microscopic cross section lookup with Doppler Broadening techniques to accelerators may contribute to a significant performance improvement, we can offload partial cross section lookup as well in order to make our implementation more adaptive to other complex cases. From programming model side, we also have interest to use other high-level programming languages such as Kokkos [15] and SYCL [16]. From performance evaluation side, more tests need to be done so as to cover a wider range of architectures. We intend to adopt several metrics [17] for the evaluation of portability and performance portability.

Acknowledgments. This work was performed using HPC resources from GENCI-IDRIS (in the framework of Cellule de Veille Technologique) and CEA-CCRT (Research and Technology Computing Center).

References

1. Brown, F. B., Martin, W. R.: Monte Carlo methods for radiation transport analysis on vector computers. Progress in Nuclear Energy, 14(3), 269–299 (1984)
2. Ozog, D., Malony, A.D., Siegel, A.R.: A Performance Analysis of SIMD Algorithms for Monte Carlo Simulations of Nuclear Reactor Cores. In: 29th IEEE International Parallel and Distributed Processing Symposium, pp. 733–742. IEEE Press, Hyderabad (2015)
3. Hamilton, S.P., Slattery, S.R., Evans, T.M.: Multigroup Monte Carlo on GPUs: Comparision of history- and event-based algorithms. Annals of Nuclear Energy. 113, 506–518 (2018)
4. Hamilton, S.R., Evans, T.M.: Continuous-energy Monte Carlo neutron transport on GPUs in the Shift code. Annals of Nuclear Energy. 128, 236–247 (2019)
5. Bergmann, R.M., Vuji, J.L.: Algorithmic choices in WARPA framework for continuous energy Monte Carlo neutron transport in general 3D geometries on GPUs. Annals of Nuclear Energy, 77, 176–193 (2015)
6. Bleile, R.C., Brantley, P.S., Dawson, S.A., O'Brien, M.J., Childs, H.: Investigation of portable event-based monte carlo transport using the nvidia thrust library (No. LLNL-CONF-681383). Lawrence Livermore National Lab. (LLNL), Livermore, CA (United States) (2016)
7. Brun, E., Chauveau, S., Malvagi, F.: PATMOS: A prototype Monte Carlo transport code to test high performance architectures, in Proc. M&C 2017, Jeju, Korea, April 16–20 (2017)
8. Tramm, J.R., Siegel, A.R., Islam, T., Schulz, M.: XSBench — The Development and Verification of a Performance Abstraction for Monte Carlo Reactor Analysis. PHYSOR 2014 — The Role of Reactor Physics toward a Sustainable Future, Kyoto (2014)
9. Cullen, D.E., Weisbin, C.R.: Extract Doppler Broadening of Tabulated Cross Sections. Nuclear Science and Engineering, 60, 3, 199–229 (1976)
10. Tramm, J.R., Siegel, A.R.: Memory bottlenecks and memory contention in multicore Monte Carlo transport codes. In SNA+ MC 2013-Joint International Conference on Supercomputing in Nuclear Applications+ Monte Carlo (p. 04208). EDP Sciences (2014).
11. Wang, Y., Brun, E., Malvagi, F., Calvin, C.: Competing energy lookup algorithms in Monte Carlo neutron transport calculations and their optimization on CPU and Intel MIC architectures. Journal of Computational Science, 20, 94–102 (2017)
12. Nvidia: CUDA Programming Guide. (2018)
13. The OpenACC Application Programming Interface, `https://www.openacc.org`
14. Hoshino, T., Maruyama, N., Matsuoka, S., Takaki, R.: CUDA vs OpenACC: Performance Case Studies with Kernel Benchmarks and a Memory-Bound CFD Application. 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, 136–143 (2013)
15. Edwards, H.C., Trott, C.R., Sunderland, D.: Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. Journal of Parallel and Distributed Computing, 74(12), 3202-3216 (2014)

16. SYCL Overview, https://www.khronos.org/sycl/
17. Pennycook, S.J., Sewall, J.D., Lee, V.W.: A metric for performance portability. arXiv preprint arXiv:1611.07409 (2016)